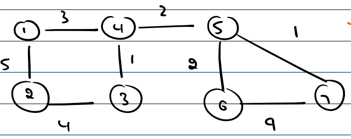
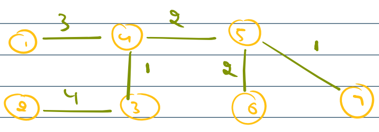
Spanning Tree



Prim's Algo
Kruskal Algo
↓
Pick the lowest possible weight edge

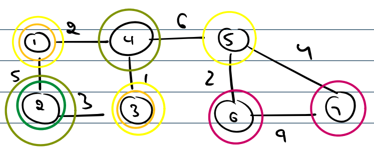


Edges ⇒ ~~1~~, ~~1~~
~~1~~, ~~2~~
~~3~~
~~4~~
5
9

Prim's Algorithm

1. remove
2. ignore if already visited → continue
3. marked visited
4. self work
5. add unvisited nbrs

x



1 4 3 2
5 6 7

visited

vtx	acqvtx	cost
~~1~~	~~1~~	~~0~~
~~2~~	~~1~~	~~5~~
~~4~~	~~1~~	~~2~~
~~3~~	~~4~~	~~1~~
~~5~~	~~4~~	~~6~~
~~2~~	~~3~~	~~3~~
~~6~~	~~5~~	~~2~~
~~7~~	~~5~~	~~4~~
~~7~~	~~6~~	~~9~~

x
x
x
x
x
x
x
x
x

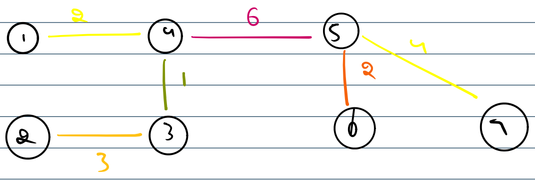
Priority Queue → cost ascending basis
↓
least cost will be popped.

self work			
1	1	@	0
4	1	@	2
3	4	@	1
2	3	@	3
5	4	@	6
6	5	@	2
7	5	@	4

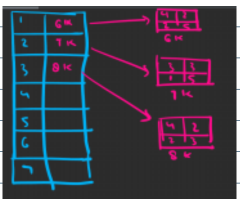
→ Answer

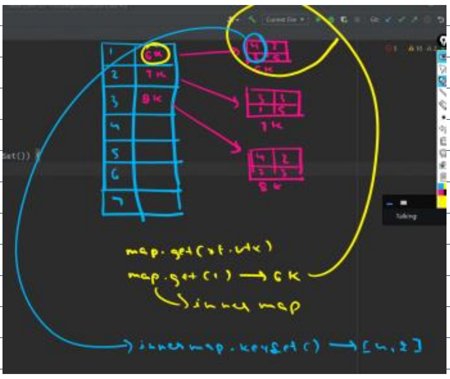
self work			
1	1	@	0
4	1	@	2
3	4	@	1
2	3	@	3
5	4	@	6
6	5	@	2
7	5	@	4

→ Answer



⇒ MST
↓
Minimum Spanning Tree



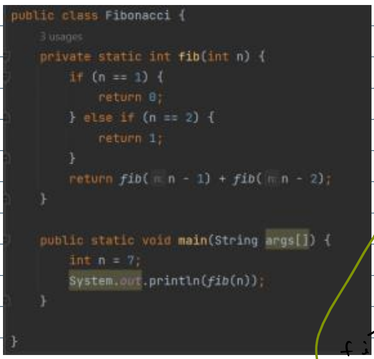


map.get(rt.vtx)
map.get(1) → 6K
inner map
innermap.keySet() → [4,2]

Fibonacci Series

1st term = 0
2nd term = 1

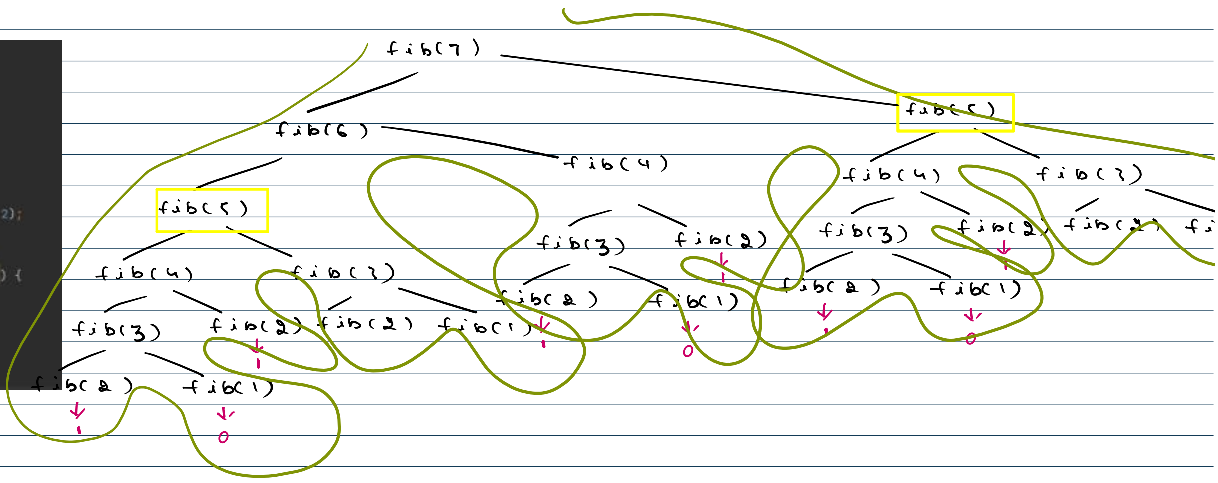
nth term = (n-1)th term + (n-2)th term.



```
public class Fibonacci {
    3 usages
    private static int fib(int n) {
        if (n == 1) {
            return 0;
        } else if (n == 2) {
            return 1;
        }
        return fib( n: n - 1) + fib( n: n - 2);
    }

    public static void main(String args[]) {
        int n = 7;
        System.out.println(fib(n));
    }

}
```



Dynamic Programming

→ Enhanced Recursion
→ memorization → memorized Recursion

When to use DP ?

Recursion + overlapping sub problems

↓

Choices
+
Decision

$$\begin{tikzcd} & \text{overlapping sub-problems} \\ \text{key properties} \arrow[ur] \arrow[dr] & \\ & \text{optimal substructure} \end{tikzcd}$$

Lookup table

It is an array or dictionary which is used to store results of solved subproblems so they don't need to be recalculated.

$$\begin{tikzcd} & \text{Top-down (memorization)} \\ \text{DP approaches} \arrow[ur] \arrow[dr] & \\ & \text{Bottom up (Tabulation)} \end{tikzcd}$$

$b(1)$